SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis, Master's Thesis, . . . in Informatics

Evaluating learning algorithms: An efficient way/Efficient ways to find Regular Inductive Statements

Author

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor’s Thesis, Master’s Thesis, … in Informatics

Evaluating learning algorithms: An efficient way/Efficient ways to find Regular Inductive Statements

Titel der Abschlussarbeit

Author:	Author
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	Submission date

Abstract

Acknowledgments

Contents

1 Introduction

As software systems grow in size and permeate more and more areas of our lives. Individuals and organizations use the majority of software in their systems. Thus the reliability and stability of the software testing are of major importance. Simulation and testing can detect bugs but not prove their absence. Such reactive systems, when no function is being computed, termination is usually undesirable. For this reason, we are interested in *property checking* or *model checking*. It has found a wide range of applications spanning from adaptive model checking.

We here consider the verification of safety properties similar to the original Regular Model Checking framework, where a program is represented using symbols and finite automata. To be more specific, our goal is to confirm that a given program cannot execute in a way that starts from a set of initial configurations ($\mathcal{I}$) and leads to a group of dedicated bad configurations ($\mathcal{B}$). These bad configurations represent conditions that should not happen during the program's execution. However, this is an undecidable question in general and tools for Regular Model Checking are necessarily incomplete. A solution to this problem was proposed in [CES09], which utilizes *inductive statements* to ensure that no undesired configuration can be reached from any initial configuration. This means that for every pair of initial and undesired configurations, there is at least one inductive statement that is satisfied by the initial configuration but not by the undesired one. By doing this, it can be concluded that no undesired configuration can be reached.

Based on automata learning, one can learn a set of inductive statements that are powerful enough to establish a given safety property. The learned language of inductive statements is a certificate of the correctness of the property. The purpose of this thesis is to collect and analyze empirical data on the performance of learning algorithms such as L*, NL*, Kearns-Vazirani, Rivest-Schapire.

Structure of the thesis

In the first part of our thesis, we consider the regular transition system and the set of all inductive statements. In the second part of the thesis,

2 Literature Review

Inductive statements for regular transition system was introduced by M.Sc. Welzel-Mohr. Motivated from paper Inductive statements for regular transition system. He used only the L* algorithm for learning the regular statements.

3 Preliminaries

In this section, we introduce some basic notions that we use throughout this thesis.

Finite automata

We use standard notions of finite automata. We distinguish between deterministic and non-deterministic automata to recognize regular languages of finite words.

Definition 3.1: *Deterministic finite automaton (DFA).*

A DFA is a quintuple $\mathcal{M} = (Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of states with a initial state $q_0 \in Q$. A set of input symbols called the alphabet Σ. A transition $\delta : Q \times \Sigma \rightarrow Q$ and a set of final states F. Let $w = a_1a_2...a_n$ be a string over the alphabet Σ. The automaton $\mathcal{M}$ accepts w if a sequence of states, $r_0, r_1, ...r_n$ exist in Q:

- $r_0 = q_0$
- $r_{i+1} = \delta(r_i, a_{i+1}), for\ i = 0, ..., n-1$
- $r_n \in F$

Definition 3.2: *Nondeterministic finite automaton (NFA).*

A NFA is a quintuple $\mathcal{N} = (Q, q_0, \Sigma, \Delta, F)$ where Q, Σ and F are as for a DFA. Let $w = a_1a_2...a_n$ be a string over the alphabet Σ. The automaton $\mathcal{N}$ accepts w if a sequence of states, $r_0, r_1, ...r_n$ exist in Q:

- $r_0 = q_0$
- $r_{i+1} \in \Delta(r_i, a_{i+1}), for\ i = 0, ..., n-1$
- $r_n \in F$

4 Inductive statements for regular transition system

In the *Regular Model Checking* framework, program configurations are represented as finite words over a pre-determined alphabet Σ. The system comprises a series of starting configurations and the transitions.

In this thesis, we will be focusing on *Regular transition system* (RTS) which uses regular languages to represent the models of the program.

4.1 Regular transition system

The *transducer* will define the behavior of the system, i.e it describes how the configurations change in the system.

Definition 4.1: *Transducer.*

A Σ-Γ-transducer is an NFA $\langle Q, Q_0, \Sigma \times \Gamma, \Delta, F\rangle$, it represents a list of pairs $\langle u_1, v_1\rangle \ldots \langle u_n, v_n\rangle$ where $\langle u_1 \ldots u_n, v_1 \ldots v_n\rangle \in \bigcup_{n\geq 0} \Sigma^n \times \Gamma^n$.

EXAMPLE FOR TRANSDUCER

Definition 4.2: *Regular transition system (RTS).*

An RTS is a triple $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$ where Σ is finite alphabet while $\mathcal{I}$ is an NFA, which represents initial configurations. And $\mathcal{T}$ is a Σ-Σ-transducer.

EXAMPLE FOR RTS

4.2 Inductive statements

Definition 4.3: *Interpretation.*

For any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$, we call a pair $\langle \Gamma, \mathcal{V}\rangle$ an Γ-interpretation where Γ is a finite alphabet and $\mathcal{V}$ is a deterministic Σ-Σ-transducer. In the following, we denote $u \models I$ to indicate $\langle u, I\rangle \in [[\mathcal{V}]]$.

Definition 4.4: *Inductive statements.*

For any given Γ*-interpretation for* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$*, we define*

$$Inductive_{\mathcal{V}}(\mathcal{R}) = \{I \in \Gamma^* | \forall u \rightsquigarrow_{\mathcal{T}} \ . if \langle u, I \rangle \in [[\mathcal{V}]] \ then \ \langle v, I \rangle \in [[\mathcal{V}]]\}$$

$$= \{I \in \Gamma^* | \forall u \rightsquigarrow_{\mathcal{T}} \ . if u \models I \ then \ v \models I\}$$

Definition 4.5: *Potential reachability.*

Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ *be any RTS and* $\langle \Gamma, \mathcal{V} \rangle$ *any interpretation. We write* $u \Rightarrow_{\mathcal{V}} v$ *if and only if* $u \models_{\mathcal{V}} v$ *for all* $I \in target_{\mathcal{V}}(u) \bigcap Inductive_{\mathcal{V}}(\mathcal{R})$. $\overset{Inductive}{\Longrightarrow}_{\mathcal{V}}$

4.3 Concrete interpretations

5 Algorithmic Learning of Finite Automata

Learning automata is a computational model for solving problems, where an agent learns to optimize its behavior by interacting with an unknown environment. The agent, also known as a learner, observes the feedback from the teacher, updates its internal state, and adjusts its actions accordingly. The *Teacher* This interaction process between the *Learner* and the *Teacher* is the primary mechanism of learning automata. In the field of automata learning, there are generally two distinct settings: active and passive learning. Passive algorithms are provided with a fixed set of examples consisting of strings that the automaton should either accept or reject. Active algorithms, unlike passive ones, have the ability to expand the set of examples as needed by asking further queries. However, in this thesis, our focus is solely on active learning. We do not introduce passive learning here but refer the interested reader to [CES09].

This chapter aims to provide a deeper understanding of the process of learning automata, including the roles and responsibilities of the *Teacher* and *Learner* in Section 5.1. In Section 5.2, we will introduce several of active algorithms that use for our experiment.

5.1 The oracles

In this learning scenario, the *Teacher* is proficient in the language being taught and is responsible for answering any questions posed by the learner. The *Learner* is given the opportunity to ask two types of queries - membership and equivalence. Membership queries are used to classify a word based on whether it belongs to the language being taught or not. Equivalence queries, on the other hand, are used to determine whether an assumed automaton is equivalent to the language the *Teacher* has in mind. The learning process continues until the *Teacher* answers an equivalence query positively.

Membership oracle The *Learner* provides a word $w \in \Sigma^*$, the *Teacher* replies "yes" or "no" depending on whether $w \in \mathcal{L}$ or not.

Equivalent oracle The *Learner* conjectures a regular language, typically given as a DFA $\mathcal{M}$, and the *Teacher* checks whether $\mathcal{M}$ is an equivalent description of the target language $\mathcal{L}$

and return "yes", otherwise return an counterexample $u \in \Sigma^*$ with $u \in \mathcal{L}(\mathcal{M}) \iff u \notin \mathcal{L}$ or $u \in \mathcal{L} \iff u \notin \mathcal{L}(\mathcal{M})$.

On equivalent oracle, the *Teacher* can return a positive counterexample or a negative counterexample. A positive counterexample is a missing word in the conjecture but present in the target. The negative one is defined similarly.

It is crucial for the *Teacher* to have a clear and specific understanding of the correct hypothesis. Since we know how to implement the *Teacher* to answer the oracles, it is now simple to apply different of learning algorithms.

5.2 Algorithms

A learning algorithm—often called learner—learns a regular target language $\mathcal{L} \subset \Sigma^*$ over an a priori fixed alphabet Σ by actively querying a teacher. We apply several of these algorithms in the course of this thesis.

5.2.1 L*

L* learning automata was introduced by Angluin in 1987.

5.2.2 NL*

In general, a nondeterministic finite automata *NFA* is often preferable to a deterministic finite automata *DFA* due to potentially exponential differences in their sizes (REFERENCE FOR COMPARISON OF NFA AND DFA). As such, learning algorithms for *NFA* are needed. In this section, we will introduce another active learning algorithm called the *NL** algorithm, patterned after *L**. The *NL** infers a canonical residual finite-state automata, a subclass of nondeterministic finite automata was introduced in the seminar work [CES09].

5.2.3 Kearns-Vazirani

5.2.4 Rivest-Schapire

6 Implementation

We apply automata learning algorithms to solve the regular model checking problems as well as finding an inductive statements for regular transition system.

Membership query On a membership oracle, the learner provides a statement and asks the teacher if this statement whether inductive or not. As we described in Chapter 4, a statement I is *inductive* if, for any transition $v \rightsquigarrow u$ where u satisfies I, u also satisfies the statement. REF TO ABOVE DEFINITION. Therefore, one can simply implement the Membership Oracle by checking the acceptance of $\mathcal{M}$ while $\mathcal{M}$ is an automaton for $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ and negating the answer. The $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ is defined by:

$$\overline{Inductive_{\mathcal{V}}(\mathcal{R})} = \{I \in \Gamma^* \mid \exists u \rightsquigarrow_{\mathcal{T}} w \,.\, u \models I \, and \, w \not\models I\}$$

Let $\mathcal{T} = \langle P, \Sigma \times \Sigma, \Delta, p_0, E\rangle$ is a transducer and $\mathcal{V} = \langle P, \Sigma \times \Gamma, \delta, q_0, F\rangle$ is an interpretation. The automaton of $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ is defined by $\langle Q \times P \times Q, \Gamma, \triangle, \langle q_o, p_0, q_o\rangle, E \times F \times (Q \setminus F)\rangle$ where

$$\triangle(\langle q_1, p, q_2\rangle, I) = \delta()$$

Algorithm 1 Membership query

```
Input: Statement $\mathcal{I}$
Output: True or False
begin
  $\mathcal{M} \leftarrow getAutomaton(\overline{Inductive_{\mathcal{V}}(\mathcal{R})})$
  if $\mathcal{I} \in \mathcal{L}(\mathcal{M})$ then
      return false;
  else
      return true;
  end if
end
```

Algorithm 2 Equivalent query

```
Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ do
      if $N$ is even then
          $X \leftarrow X \times X$
          $N \leftarrow \frac{N}{2}$                                ▹ This is a comment
      else if $N$ is odd then
          $y \leftarrow y \times X$
          $N \leftarrow N - 1$
      end if
  end while
```

7 Experiments

8 Conclusion

We studied Regular Model Checking of safety properties. We evaluated the performance of our algorithms based on a prototype implementation.

Open Questions and Future Research

Abbreviations

List of Figures

List of Algorithms

List of Tables

Bibliography

[CES09] E. M. Clarke, E. A. Emerson, and J. Sifakis. “Model checking: algorithmic verification and debugging.” In: *Communications of the ACM* 52.11 (2009), pp. 74–84.